Run-time Configurable GUIs in Java

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

Sunil P I

to the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

January 1998

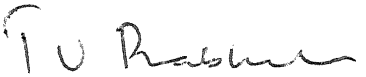
CSE-1988-M-SUN-RUN

Entered in systems

NJSN
6-4-95

CERTIFICATE

This is to certify that the work contained in the thesis entitled Run-time Configurable GUIs in Java by Sunil P I has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.



T V Prabhakar,
Professor,
Department of Computer Science & Engineering,
Indian Institute of Technology, Kanpur.

Abstract

The importance of the user interface in software forces the developer to spend a considerable majority of the development time on optimizing the interface. Even an optimum interface does not please all the users, though it addresses the majority.

The typical solution suggested was, allowing the user to customize and configure their interface. Many interface libraries have come up, providing rich sets of configurable interface components, implementing this solution.

The disadvantages associated with this approach are, the significant amount of effort which should be taken by the developer, the non-conformance of the look and feel of the new components to the existing standards, and the difficulty involved in converting an existing program to use the new set of components.

This thesis suggests a solution for avoiding the above disadvantages by providing customizability, configurability and dynamism to the interface, while sticking on to a set of interface components, which the developers and users both are familiar with. The Java AWT components have been selected here for implementation.

The `Customizer` implemented in this thesis, allows user to change the layout, change the properties of components, substitute one component with another, add or delete components etc., at run-time. `Customizer` is implemented in such a manner that, the developer can use it with the least overhead.

Acknowledgments

I would like to express my deep sense of gratitude to my thesis supervisor, Dr. T. V. Prabhakar for his expert guidance and encouragement throughout this thesis work. Thanks to him for introducing me to the field of Human-Computer Interaction, the Java programming language and this wonderful idea to implement. I am also thankful to him for having gone through this report and structuring it in the present form.

I am thankful to the lab staff of Department of Computer Science and Engineering and Computer Center for their assistance and cooperation during this work. I would like to specifically thank Mr. Anil Kommuri, Flt. Lt. Devesh Singh and Mr. Sharad Gang for their technical help.

I would like to thank my wife for her patience and support from away and near during my entire course. I would like to thank Sq. Lr. Vinu Thomas, Mr. Gopi N Pillai & family, Mr. Jamal Jafar & family for making my life here pleasant.

Finally, I would like to thank all my friends at IIT K, who made my stay here an enjoyable and memorable experience.

Contents

List of Figures

Chapter 1

Introduction

1.1 Graphical User Interfaces

1.1.1 Introduction

One of the most important developments in information technology over the past ten years or so - quite apart from the massive improvements in hardware technology - has been the graphical user interface (GUI).

The user interface is critical to the success of any application. To the users, the user interface is the application. An application may support every conceivable function, but if it does not meet the users needs, it will not be perceived as successful. Good user interfaces are created by following an iterative design process that involves the user, and by following proven design principles.

The advent of the microcomputer had a profound effect on information technology. Now that computers were being mass produced, the more units that could be sold the cheaper they would be - and the cheaper they were the more people could afford to buy them. Cost, however, is not the only barrier to the purchase of computers – they also had to be usable. So it was effectively with the advent of the microcomputer in the early 1980's that the discipline of human-computer interaction (HCI) became established.

Undoubtedly the greatest innovation in the field of HCI has been the adoption of the graphical user interface (GUI). For the users, the graphical user interface is

much more usable than anything that has come before.

1.1.2 Origin

The origin of graphical user interfaces (GUIs) can be traced back to the 1970's [4]. Research in this area began at the Xerox Parc by starting the project SmallTalk, to probe the question: In the future, computing power would be abundant and inexpensive. How could the best use be made of the power available? Two influential developments resulted: object-oriented programming and the graphical user interface.

By pursuing the latter idea, Star GUI was implemented on the Alto system and included most of the elements that are so familiar today: windows, icons, menus, etc.(Smith, Irby et al., 1993) It was Apple, who went on to commercialize this idea, rather than Xerox Parc, under the initiative of Steve Jobs. They designed the new Apple system, named Lisa, but it didn't succeed commercially.

Unperturbed, Apple continued their work and Macintosh was released in 1984. But, for its time it was under-powered and over-priced, compared to available PC clones. What it had that was different was its user interface. At that time the style of interface was usually referred to as Wimp, derived from the components window, icon, menus and pointer, but that term seemed to die out due to its negative connotations and was replaced by GUI - or even gooie.

1.1.3 Relevance

The reasons why GUIs are easier to use are many and have been explored and documented in many research papers and books(Chapter 4 of Dix, Finlay et al., 1993). To most of the users the question never arises. Most people who have tried a GUI have found it much easier to use than the alternatives like command language. (The general exception to this is expert users of traditional interfaces who often find GUIs limiting and long-winded). The components of the interface similarly need no description to people, they can simply be given to them.

In the earlier days, a new software came only once in a while . In that case,

the users get sufficient time to get accustomed to it, however crude the interface is [The users were forced to do so, because there were no alternatives]. But, with the present day scenario of scores of software systems coming from different vendors for the same purpose, the user is most likely to select the one with the most attractive interface. Also the life time of an application has significantly reduced. Hence the argument of the expert users does not hold.

1.1.4 Current Limitations

Modern software development techniques lack effective methods for building good user interfaces. Today's ad hoc methods require a significant proportion of total development time, while the resulting interfaces tend to fall short in key areas /citecust-ui.

There are scores of GUI development environments in use today, and many of them provide a rich set of interface components. But the GUI once produced remains static throughout the life of the software. The GUI, whose aim is to make the HCI most convenient, is determined by the developer, instead of empowering the users to decide their interface.

This forces the developer to spent a significant amount of time in designing the GUI of the software, rather than concentrating on the functionality of the application. Many software systems have been discarded only because of poor interfaces. Therefore the scenario is, the developer takes much pain in optimizing the GUI but not satisfied because what type of user is going to interact is not a priori known.

1.2 Desirable Properties of a GUI

A good GUI must possess certain characteristics [7]. In addition to them, the following features are discussed here.

- Configurability
- Customizability
- Dynamism

- Uniformity

1.2.1 Configurability

A configurable GUI is one which allows the user to configure it at any time, essentially while running. The configuration of a GUI refers to the layout of controls in the interface. The layout of controls are decided by the position, shape and size of the components in the GUI.

Features

The layout of the GUI is determined by the position and size of the interface components in it. Usually the developer decides the optimum values for those parameters and places the components appropriately.

Each user may have their own tastes for the GUI layout because, each of them will use the interface in their own way. For example, one user may prefer two components to be close together while another user may prefer them to be at two different corners.

Every user will prefer large size for the components, which are used frequently. Such tastes of the users demand different layouts from the developer.

For example, a developer may decide that a button will be used frequently and he may optimize the size of that by following a typical guideline as given below.

The typical size of a Button(Study by Hall, Cunningham, Roache and Cox, 1988) is recommended to be as, 27 mm wide and 22 mm high.

According to the guideline, the developer may determine the optimum size of a Button. But one user may not use that Button at all, who will want the size of that Button minimized. One other user who may rely heavily on that button may want it's size to be increased.

The developer cannot produce GUI's for all sort of users. The only option which is viable is, to give the user the power to decide the interface.

▮ Advantages

Providing a configurable GUI to the user not only pleases every user, but also reduces the development time for the GUI. Since the GUI is configurable, the GUI determined by the developer no longer becomes critical and thus the developer can afford to shift the attention more towards the functionality of the software. This results in the production of improved software systems.

1.2.2 Customizability

A good user interface should be customizable. A customizable interface allows user to change the manner in which she or he interacts with the application.

Customizability refers to the number of things over which a user has control [6]. Customizability is an important factor in the usefulness of an application, since a highly customizable application allows its user to 'teach' the application the manner in which the user wants to use it. Such applications better meet users needs by not forcing the user to interact 'its way.'

▮ Features

The way in which a component behaves in the GUI can be changed by changing the properties of the component or by replacing the component with an alternate component. Providing both these facilities to the end-user makes the GUI tremendously powerful.

The properties of a component decides the look and feel of it. The user can be given the option to change the color/font of the component. Also the facility to rearrange the options in a selection menu or renaming them can be given.

Yet another powerful tool at the hands of the end-user will be to substitute an existing component with an alternate one. Even though the possible substitutions for a component are to be decided a priori, this gives vast amount of flexibility to the end-user.

Advantages

Presently, for the interaction with the user the developer has to decide on a single component and that component is fixed throughout the life of the application. But with this substitution facility, it will be one from a set of similar components and the user can switch between any of the other components in the set.

Also, by changing the properties of the component, the users can get the look and feel which they desire. For a component presenting multiple options, the user may decide to change the order/name of the options to suit their selections in a more comfortable manner.

1.2.3 Dynamic GUI

Even in a GUI, which is Customizable and Configurable, the total number of components will remain the same, as also their purposes. ie, we can say that the GUI is static regarding the presence or absence of components.

A dynamic GUI allows the user to add or delete components at run time. The addition of components can be applied to static components only. The deletion of components can be applied to any of the components at any time the user feels that it is no longer necessary.

Features

Any component can be deleted at any time whenever the user feels that it is no longer necessary. This may enable the user to interact with the GUI in a better manner, because he doesn't have to keep the controls he is not going to use. This may result in a better interface.

Components like clock can be added at run time to make a better interface. The addition of static components like label may be used by the user to enhance the interface like adding description to any of the already existing components.

Advantages

By keeping only those components, which the user wants to interact with, on the interface allows the user to keep an organized and spacious interface, instead of an interface with too many controls, which the user is not going to interact with.

The advantages resulting from the addition of a new component like clock is self-explanatory. New components like labels can be added to the existing interface to make the purpose of any other component more evident.

1.2.4 Uniformity

A uniform interface provides the user with the same look-and-feel [6]. Uniformity applies to interactions within a given application and to interactions throughout a set of applications. Usually, a uniform interface conforms to some look-and-feel standard. It allows experience with one application to be applied to a new application.

For example, a user who is accustomed to entering 'q' to quit an applications, would like to enter 'q' to quit a new application rather than, for example, 'e' to exit the new application. New applications are easier to learn if they share common commands with existing applications. The same is true for learning a new feature of an existing application. If the commands of the new feature are the same as the commands of a previously known feature, the new feature will be easier to learn.

Features

Using the same type of menu for accepting a file name from the user will facilitate better and fast interaction by the user. Once the user get accustomed to certain type of controls, he may not like to use any other control for that purpose.

Advantages

The application of uniformity to the GUI's results in the following advantages [5].

- Reduces the memory load to the user

If all the GUI's follow the same convention, the user will have the minimum details to remember.

- Reduces errors during interaction

  It is observed that the number of errors in the interaction will be reduced, if we conform to the uniformity principle. Errors, if any, in an interaction with a GUI results mostly from mistaking one control for another. This possibility is completely eliminated by following the uniformity principle.

- Reduces training time

  A GUI following the uniformity principle demands minimum time from a new user to get accustomed to it.

1.3 Motivation

1.3.1 Developer's View

Now a days, the developer of an interface for a software spends an enormous amount of time in optimizing[1] the GUI. This has become mandatory, since, once the interface is decided and the software becomes a product, the GUI remains static or stationary.

Even if one optimizes the GUI, it may not suite the taste of every end-user. An optimized GUI satisfies majority of the users, but there will be always a minority, who may not be pleased with that.

This scenario arises from the following reasons:

- The power of determining the GUI lies with the developer.

- The end-users are not given the power to modify or customize the GUI.

[1] Optimizing a GUI refers to finding the appropriate controls, deciding a good layout, setting their properties etc. towards achieving the best interaction with the user.

1.3.2 End-user's View

Each end-user will have their own tastes and likes depending on several factors like

- Familiarity with certain software specific GUI's.
- Familiarity with certain input devices.
- Capabilities of the output/input devices.
- Human factors like dislike or likeness to certain controls, mood and patience of the person, age group of the person etc.

Usually a software will give you only one interface and the user has to live with that even if the person hates the interface .

1.3.3 Background

Providing the power of customizability and configurability to the end-user is not a new concept as such. Many GUI libraries have been coming up with a rich set of such configurable components. Chapter 2 contains a brief survey of such libraries.

It is found that all those interface libraries, which provide a set of configurable components, have their own components. That is, those interface components are not specific to any of the existing standard GUI. That is, an interface developed from such a library will have there own look and feel and this may go against the much desired property of the uniform look and feel of interface components.

Another disadvantage with this approach is that it demands a significant amount of effort from the developer. Because, to use that, one has to familiarize with the functionality and properties of those components .

1.3.4 Our Proposal

The proposal is to incorporate configurability and customizability to an interface library, which the users and developers are already familiar with. To the developer this view will not add any burden because they will continue to use the same set of components. The event handling mechanism also remains the same.

Thus, while sticking to the same set of interface components with the same properties, the power of configurability and customizability is added, thus making the whole system tremendously powerful.

Most of the benefits from using this tool goes to the end-user as, they can decide their own interface while adding no burden at all to the developer. Section 2.6 contains a brief comparison of this approach with the state of the art.

Chapter 2 does a survey of the existing technology relating to this concept. Chapter 3 describes the tool implemented in this thesis. Chapter 4 gives the design and implementation details.

Chapter 2

Configurable GUIs

2.1 Overview

Nowadays, lot of component libraries have been coming up from various vendors. These interface libraries are considerably rich in the abundance of components. These components also provide enough flexibility to the user for customization and configuration.

All these people have adopted the general strategy of developing their own components. This technique offered them complete control over these components. This way, they are able to offer a good amount of flexibility in redesigning the interface.

But this approach introduces certain disadvantages. Since these components are their own creations, they will have their own properties. Also the way of using them, the event handling mechanism and other properties may be completely new as far as the developer is concerned. This demands some effort from the developer, for getting accustomed to this new interface library.

Also, the conversion of an existing program to use any of these interface libraries will involve a great deal of work. One other argument against this approach is, these new components may have a look and feel which is different from any of the existing standard libraries. This may go against the desired properties for a good GUI like uniformity and consistency.

A typical example of such a library is SUIT

2.2 Suit

2.2.1 Introduction

SUIT (Simple User Interface Toolkit) [1] is a library of interface tools developed at the University of Virginia. This is meant for C programmers to create mouse based interfaces, which shall be configurable by the end-user.

2.2.2 Features

- Widget types

The following are the interface widgets offered by SUIT.

1. A label widget for displaying text
2. A bounded value widget.
3. A polygon widget (used mostly for demonstration purposes)
4. A menu widget
5. A set of radio buttons for mutually exclusive choices
6. A Text editor
7. A scrollable list of text for selecting words/lines of text from a list.
8. An on/off switch
9. A type-in box for letting users specify strings
10. A set of color chips for selecting colors
11. A set of buttons to exit the program.

- Changing the Layout

SUIT allows the end-user to move and resize widgets while the application is running.

Moving widgets By holding down the SHIFT and ALT keys, one can drag and drop any component, while running the program.

Resizing Widgets In addition to moving widgets, you can also resize them. For resizing a widget, user has to do the above operation on the desired corner of the widget.

- Changing Properties

Each SUIT widget maintains information like color, font etc. in the form of a collection of variables or properties that govern the widget's appearance and functionality. The properties of an SUIT widget can be viewed or changed by invoking the SUIT property editor.

- Changing Implementation

Each SUIT widget may have one or more display styles and the end-user may cycle between the different display styles by pressing the character 'c' after pointing the mouse to the required widget.

- Adding new components

SUIT allows addition of widgets to an application on the fly. The type of widgets that can be added include clock, label etc. A new widget can be added by pressing 'n' while holding down SHIFT and ALT keys.

2.3 Sub-Arctic

2.3.1 Introduction

SubArctic [2] is a new Java-based user interface toolkit which offers features like animation and graphics support for a set of interface components. SubArctic has

been developed by Scott Hudson and Ian Smith at the Graphics Visualization and Usability Center at Georgia Tech.

2.3.2 Features

SubArctic provides all of the basic interactors (widgets) for building traditional, static two-dimensional interfaces. It provides a complete library of interactive objects including buttons, check-boxes, radio-buttons, scrollbars, labels, menus (including pop-up menus), menubars, icons, image buttons etc. All of the standard UI components are currently implemented using a Motif-like look − and − feel [9] but this can be changed by the programmer;

Various sorts of animations and graphical operations can also be done over these interface components.

2.4 Empress GUI

The EMPRESS GUI BUILDER [3] provides an object-oriented development environment for creating aesthetically pleasing, interactive and graphical applications. Developers can use its mouse-driven point-and-click tools to give applications whatever look and feel they desire within the Motif framework [9].

EMPRESS was designed by John Kornatowski and Ivor Ladd. at Empress Software Inc., Toronto.

2.4.1 Features

Developers can manipulate the size, scale color, function and placement of interface objects with the same point-and-click ease as desktop drawing software. Plus, they can create, name and apply visual styles of their choosing to give their applications a consistent and standardized look. Objects can inherit styles, greatly simplifying development.

EMPRESS GUI BUILDER provides a full selection of interface objects such as buttons, windows, image and audio fields, hypertext boxes, labels and separators.

2.5 Disadvantages

The interface libraries quoted above are typical examples of the commercial interface libraries available today. We are able to observe that, all those have developed their own components. While SUIT components have their own look and feel, SubArctic and EMPRESS have adopted the standard look and feel of Motif.

To use any of the above interface libraries, one has to become accustomed to the new set of components offered. This may not be as easy as claimed by these people and developers will be reluctant in switching over to the new set of components.

There involves a lot of work in changing an existing program to use any of these interface components, due to the above stated reasons.

2.6 Comparison with our tool

Our tool differs from the commercial interface libraries in the above mentioned aspect. This tool does not define a set of new interface components or either adds any new component. Rather, it sticks on to the same set of components provided by Java AWT, which an ordinary Java programmer will be quite familiar with. Also, neither the properties of the components nor the event handling mechanism of the AWT has been fiddled with.

This strategy offers the least resistance to the Java developers to use this tool, because absolutely no additional burden is associated with using this. Also, none of these vendors are providing facility to remove any of the components.

Also, while the modification of an existing program to use our tool can be easily done, it involves a lot of drudgery to use any of the above mentioned interface libraries.

Chapter 3

The Tool - Customizer

3.1 Overview

The Java language has been gaining much popularity nowadays and its AWT package provides a rich set of interface building tools. For demonstrating the idea suggested by this thesis, we have selected the Java AWT for implementation. Any Java programmer who is familiar with the AWT components can make use of this tool with the minimum effort. Appendix A contains manual for developer and appendix B contains user documentation.

3.1.1 Java AWT

The AWT [11] (Abstract Window Toolkit) package of Java language gives the user a powerful environment for managing windows. It contains numerous classes and methods for designing a full fledged interface. It basically provides the standard GUI components like Buttons, Lists, Menus etc. and high level components like Windows, Menu bars, Frames, Dialogs etc. Refer Fig 1.

3.1.2 Container

Container is an abstract subclass of Component, which can hold other components. Panel, Window, Applet, Dialog, Frame, FileDialog etc. are subclasses of the

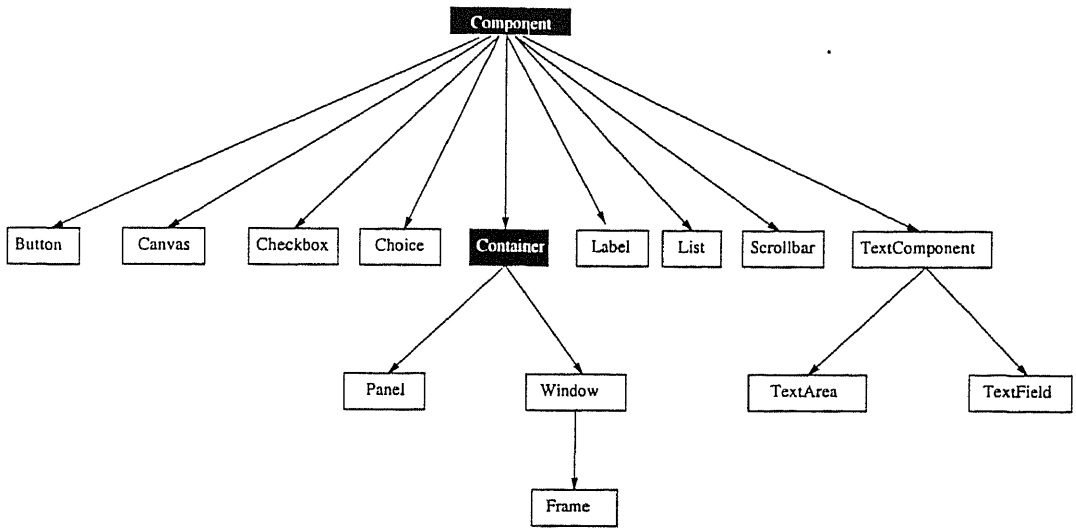


Figure 1: Hierarchy of relevant AWT classes

container.

Whenever we present components on the screen, they have to be within a container. A container can contain another container, because container itself is a subclass of component.

3.1.3 Components

The components in the AWT package refer to controls like `Button`, `Canvas`, `Checkbox`, `Choice`, `Label`, `List`, `Menu`, `Menubar`, `Scrollbar`, `TextArea` and `TextField`. These components are meant to display or represent certain information on the screen and to accept inputs from the user. User actions on components generate appropriate events, which should be handled by the back end functionality of the program.

3.1.4 Events

Event class [11] is a platform independent class defined in the AWT package to encapsulate user generated actions from the local GUI platform. Every valid action from the user on a component generates an appropriate event.

Every event object will represent the following information about the user action, among other informations.

- target

  The component which received the user action.

- when

  The time of user action.

- x,y coordinates

  The x,y coordinate position on the container where the action occurred.

- id - action identifier

  The integer code of the actions as predefined in the Event class.

3.1.5 Event handling mechanism

The AWT is built on top of native toolkits [8]. For example, in a Windows 95 environment, all the Java components are implemented using Windows objects. Every user action generates a native event [8] and is handled by the native code. This code is responsible for changing the appearance of the component (eg: When we click a button, a depressed feeling is simulated). This is the first layer of event handling code.

AWT supplies a layer of event handling code that will be called on a native event occurrence, which is also native (written in C and different for each toolkit). This layer is responsible for calling the Java interpreter using the AWT component's peer object. The peer's `action()` method prepares an AWT event from the information passed on to it and calls the `postEvent()` method for the object.

The `postEvent()` [11] method eventually calls `handleEvent()` [11] method of the component, its parent, its grand parent and so on until one of the methods return true. If none of these methods true, then the `handleEvent()` of the peer object is called. These methods are also native.

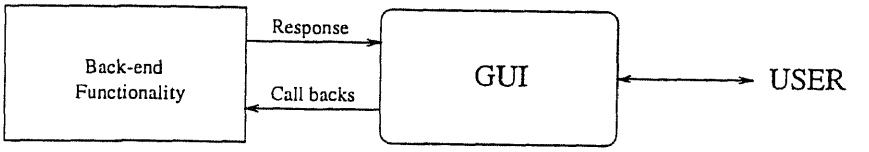


Figure 2: The Normal interaction between a software and the User

3.1.6 Layout Manager

The layout manager [10] is responsible for arranging the components in a container according to a specific algorithm. The layout managers in the AWT package are `Border`, `Card`, `Flow`, `Grid` and `Gridbag` layouts [11].

These managers use their own algorithm to position the controls on the container and vary themselves regarding the extend of flexibility offered. These managers take away the burden of positioning the controls manually on the screen from the developer.

3.2 Customizer

The `Customizer` will act as an intermediate layer between the back-end functionality and the user. The normal communication between the user and the back-end functionality may be represented as shown in Figure 2.

The software using `Customizer` will pass on the original GUI to it. `Customizer` maintains an enhanced GUI[1] for the software after reading out all the components in the original GUI. The responses from the GUI initiated by the user will reach `Customizer` first. The valid responses will be redirected to the software itself after making necessary changes in the response generated. Figure 3 shows the new architecture.

[1]The original GUI along with the additional controls for changing the layout

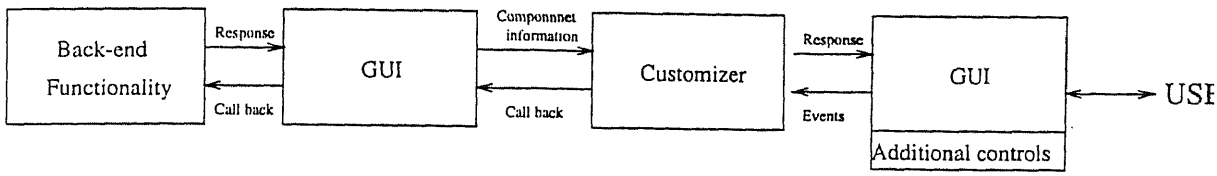


Figure 3: The new architecture on using Customizer

3.2.1 Dummy interface

The actual interface generated by the developer will not be displayed to the user. Instead, Customizer will read information about all the components in the original interface to make a dummy interface and displays that to the user.

The container as such is available here by the call to Customizer from the original software . Once the container is available, all the components and their characteristics can be deciphered and an interface can be generated accordingly. The user finally interacts with this dummy interface only.

The dummy interface uses the Dialog Layout manager [12] to achieve pixel level control. The dummy interface also provides two additional buttons to go back to the original interface and to change the layout. They will be called the DefaultInterface button and the ChangeLayout button respectively.

These buttons will be shown at the bottom of the screen in a peculiar background to distinguish their purpose from the other controls in the interface. This is illustrated in Figure 4.

3.2.2 Dialog Layout

This layout manager comes along with the Resource Wizard [12] of VisualJ ++ environment from Microsoft. This manager gives pixel level control for the components, which enables one to specify the starting position of the component along with the size of the bounding rectangle, in terms of pixels.

The other Layout managers coming along with the standard JDK package do not allow you to exercise pixel level control. When using these layout managers,

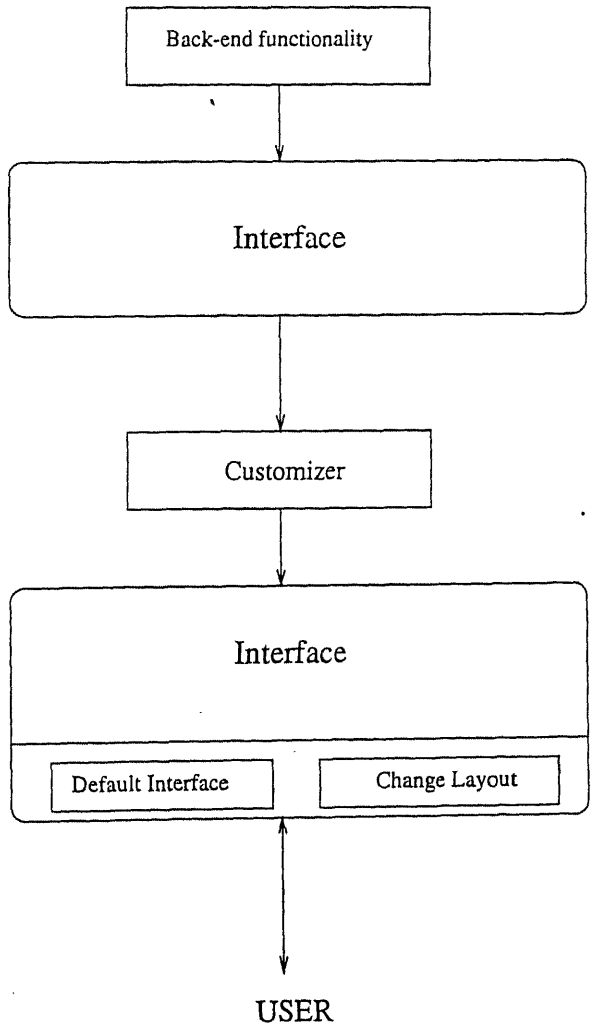


Figure 4: The dummy interface

we can specify only the appropriate size and position. The rest will be done by the layout manager. Although this view helps to relieve the programmer from the burden of positioning each and every component manually on the screen, we require pixel layer control over the components. Hence Dialog Layout manager is selected for our tool.

3.2.3 Transforming Events

The Customizer reads out all the original components, makes a temporary copy of all the components and passes the copy to be displayed by the Dialog Layout manager. This is mandatory because, the Dialog Layout manager changes some of the attributes, like starting position, and in an unpredictable way also.

Thus the components in the interface which are shown to the end-user are not the original components, but only a copy of them. Therefore, the events generated in response to user actions will be targeted to these copies and they have to be redirected to the original components.

A valid action on a component by the user may cause the following two things to happen.

An event may be generated representing that action from the native GUI platform, which will reach the Customizer first. This event is to be propagated back to the original container, because the code which handles that event lies there.

Also, in some cases, an action may make a change in one or more fields of the component. This may or may not result in a change in the look and feel of the component. eg: A mouse click on a Checkbox toggles the OFF/ON state of the component. Before passing on the event meant for a component to the back-end, the changes happened in the copy should be reflected in the original component.

3.2.4 Connecting to the back-end

The events generated automatically by the Run time system will be available to the dummy interface displayed by Customizer. Many of these events are supposed to initiate certain operations at the back-end. This can happen only if these events

reach the back-end appropriately.

Since the container of the original interface is available to Customizer, all valid events can be propagated back to it. But the events now generated will be targeted to the dummy components. The events will be modified so that it will be targeted towards the original components. This is accomplished by calling the handleEvent() of the original container with this Event object that is to be handled.

3.3 Capabilities

3.3.1 Introduction

After reading out the components from the given container and displaying the dummy interface to the end-user, Customizer takes over the control of the interface. What ever actions are carried out by the end-user, they will be first received by Customizer. Customizer, in turn sends the event generated for that action, after making appropriate changes in the event object to the back end, as mentioned before.

This will be the scenario if the user action is on any of the components in the original interface. The actions on the extra controls introduced by Customizer, as described in section 3.2.1, will open up the whole functionality of Customizer to the end-user to help redesign a new interface from the original one.

A ChangeLayout Button event creates a static representation of the present state of the interface, in the ChangeLayout window. After making some changes to the interface, the user can apply those changes by clicking the Apply button or discard them by clicking the Cancel button.

3.3.2 Changing the position

The ChangeLayout window lists the static copies of the current components. A component will be selected as the current one if the mouse cursor falls within the touch area of that component. The current component shall be indicated by highlighting with blue color. The selected component can be dragged and dropped

anywhere within the window.

3.3.3 Changing the size

If the mouse cursor falls within the vicinity of any of the corners of a component, a handle will be displayed for that corner. Dragging that handle resizes the component.

3.3.4 Changing properties

Clicking the right or middle mouse button within the touch area of a component gives the facility to change either the implementation or the properties of that component. The `substitute` option gives the facility for changing the implementation of that component (Refer 3.3.5). The `Properties` option gives access to the properties of that component. The various properties of the components are described below.

- `Color`

For every component, foreground color and background color can be specified. If the color of a component hasn't been changed previously, it will be showing off the default colors. Both the color settings[2] can be changed by selecting from a Choice control, listing the possible colors.

- `Font`

Many of the controls may contain some text. The name, type and size of the font that should be used for that component can be changed by selecting from Choice controls. After a selection is made, a new `Font` [11] object will be created with the new values and it will be used for that component further.

[2]But in the Windows '95 implementation of the native toolkit, the color attributes are not implemented for all the components.

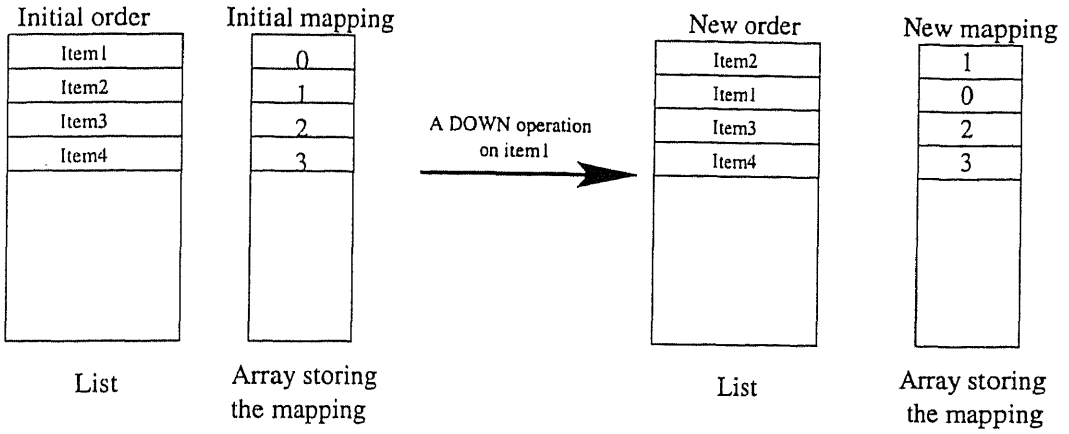


Figure 5: The illustration of the maintenance of mapping.

Label

Some of the components like Button, Label, Checkbox etc. have some label over them to identify their purpose. If the users want to change it, they are free to do that from the Properties window.

Order of options

Some of the composite components like List, Choice etc. will be listing a number of options to the user for making a selection. The order will usually be decided by the developer and the user may not like it, especially if the number of options are large. The user will be able to make changes in the order, by using the buttons Up and Down in the Properties window.

Storing the mapping To retain the same back end functionality, even after fiddling with the order of options, Customizer keeps an array for each such component, which will be used for generating the correct event. Any change in the order of items in the control will induce a corresponding change in the array to nullify that change. See Figure 5.

Generating correct events The array generated, if the user makes any change in the order, will be used at the time of event transformation, to decide on the correct item.

- Name of options

In a composite item like List, besides changing the order of items, the user is free to change the name of items also. The changed names will be shown to the user in the interface, but the original name will be used at the time of event transformation.

3.3.5 Substituting one component with another

The purpose of one component can be served by one or more other components in certain cases. If one prefers a component(s) over the others, the user is free to make that change also. As of present, we cannot say that the AWT provides a rich set of components, which limits the possible alternative components to a minimum.

- Compatible components

A set of components can be said to be compatible, if they can serve the same purpose, even if their look and feel are different. The Customizer gives the user the option to replace a component with a compatible one, if there are any.

- Possible substitutions

The current sets of compatibles can be listed as:

- Button, Checkbox.
- List, Choice.

3.3.6 Addition of new components

The user may want to add static components like label to the interface for further tailoring the interface to suit their taste. Or, the user may want to undelete an

already deleted component. Also, the user may add a wall paper to the background of the interface to make it more aesthetic. Selection of the Add Component option from the ChangeLayout window gives the user facility to do the above operations.

3.3.7 Deletion of components

The user may not want to keep the components on screen, which they don't want to interact with. Such components will only serve the purpose of cluttering up the screen unnecessarily. The user can be given the facility to clean up those controls from the screen whenever they feel to do so.

3.4 Saving the new layout

On an execution of the software using Customizer, the user may make some modifications to the interface. Naturally, they will expect that new interface to come up on the next execution.

This is accomplished by saving the current status of the components into a file each time the user makes any change to the present interface and applies it.

3.5 Reading the stored layout

Each time when the software is invoked, Customizer will look for a special file, where the status of the components in the previous invocation are stored. If that file is present, Customizer reads out the component information from it and generates the interface, presenting those components.

If such a file is not present, Customizer assumes that user has so far made no change to the interface. In this case Customizer will read out the original components themselves and displays the interface for them.

Chapter 4

Implementation

4.1 Functionality

Starting up sequence On the last invocation of the software, if the user has made any change to the GUI, that information would have been stored in a file. The DynShow class uses the FileAccess class to see if such a file exists. If that file is present, the components information is read from it, otherwise the original components information from the software will be used, for constructing the GUI. The DynShow class passes control to the ChangeLayout class if the user wants to change the layout. The ChangeLayout class allows to change the layout and returns the changed container back to the DynShow. Figure 6 illustrates the interaction between the concerned classes.

Normal interaction by the user Figure 7 represents the abstract view of the functionality of the system in the normal working mode. ie., when the user is interacting with the software in the normal manner. As shown in the figure, this tool acts as an intermediate layer between the software and the user, redirecting the events generated by the user to the back-end software. The class DynShow does all these event filtering and it uses Showdgui class to display the interface with the help of DialogLayout class.

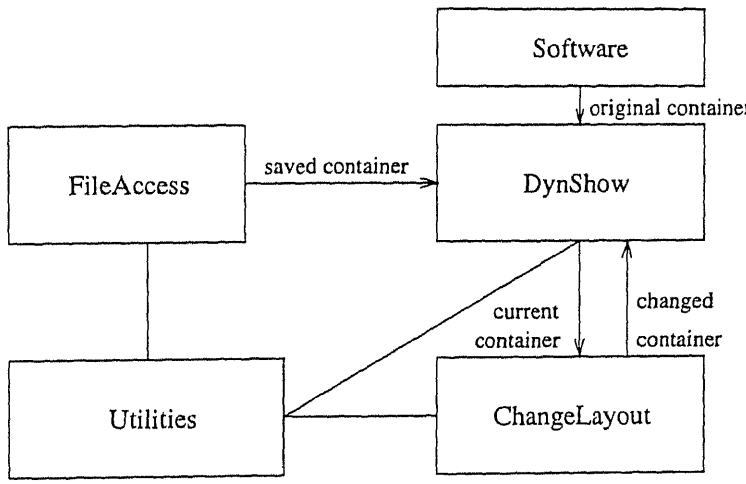


Figure 6: Starting up of the software

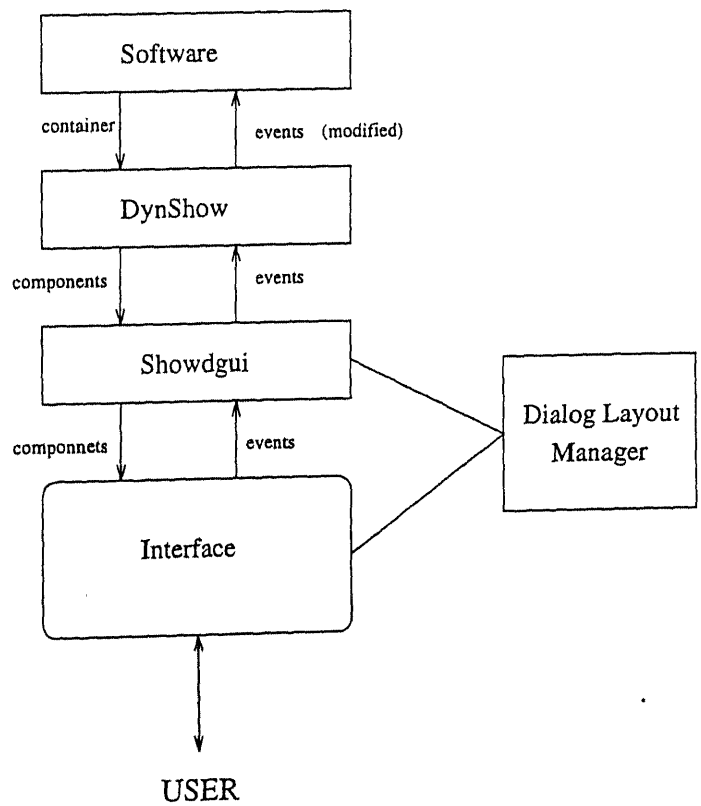


Figure 7: Normal interaction with the interface

Changing the interface The ChangeLayout class contains the functionality for changing the layout. The current component will be passed on from it to ChangeImplementation class for changes. If the change involves a substitution. ChangeImplementation handles it and gives back the new component. If the change involves any of the properties of the component, the component is passed on from the ChangeImplementation class to the ChangeProperties class. From there the appropriate PropertyChangeWindow class will be called, depending on the type of the component. The changes in the layout will be stored by calling FileSave class. Figure 8 illustrates this functionality. Any change in position or size of the components is handled within the ChangeLayout class itself.

4.2 Hierarchy of classes

The important hierarchy relationships are explained here. The hierarchy of all the classes used in this tool is given in Appendix C.

PropertyChangeWindows for Button, Checkbox and Label The windows displayed for changing the properties of the components Button, Checkbox and Label are implemented by the Button, Checkbox and Label ChangePropertyWindow respectively. These are subclasses of the abstract class BCLPropertyChangeWindow and the method applyNewProperties is overridden by them. These classes allow to change the properties of the concerned component. Figure 9 illustrates the relationship between the concerned classes.

PropertyChangeWindows for List and Checkbox ListPropertyChangeWindow and ChoicePropertyChangeWindow are responsible for providing the facility to change the properties of List and Choice components respectively. They are subclasses of the abstract class ListChoicePropertyChangeWindow as shown in Figure 10.

PropertyChangeWindows for TextArea and TextField The classes, TAPropertyChangeWindow and TFPropertyChangeWindow offer the functionality to change

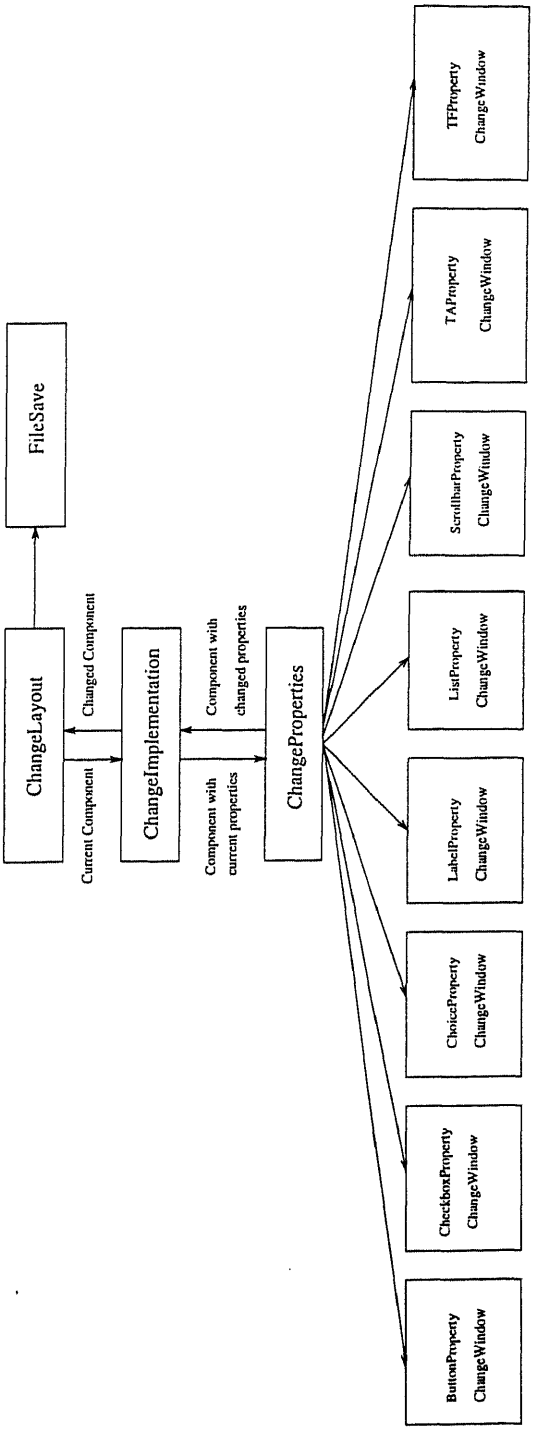


Figure 8: The functionality of `ChangeLayout` class

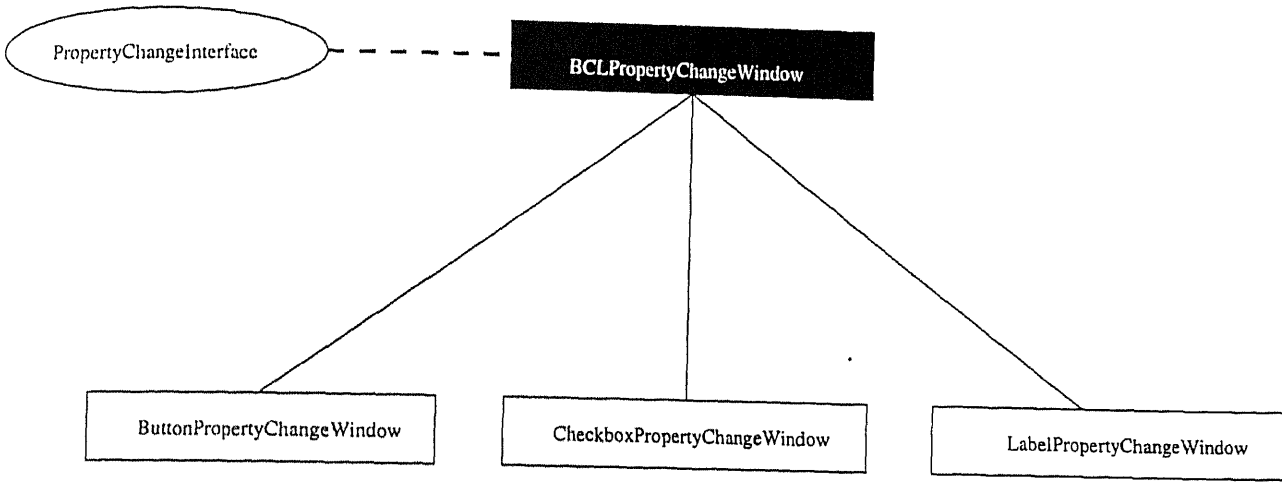


Figure 9: Hierarchy of PropertyChangeWindow classes for Button, Checkbox and Label

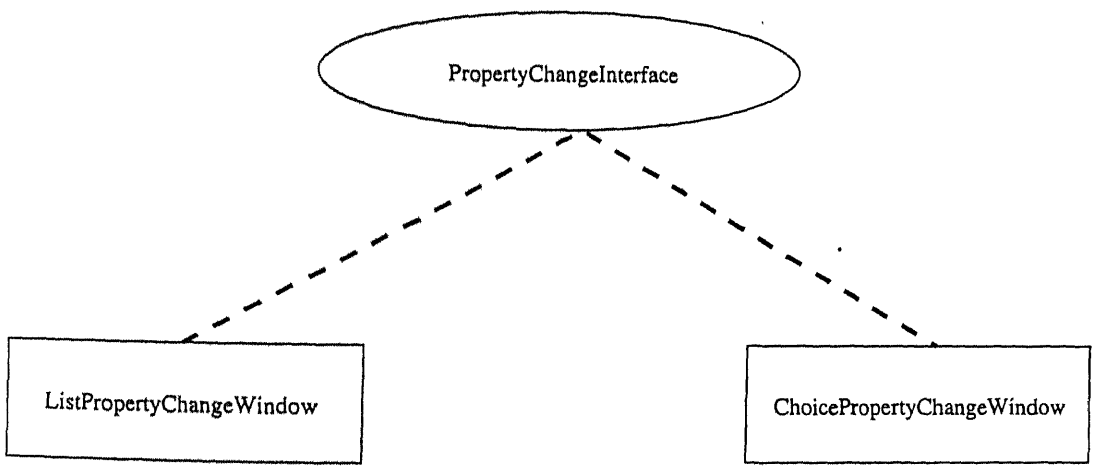


Figure 10: Hierarchy of PropertyChangeWindow classes for List and Choice

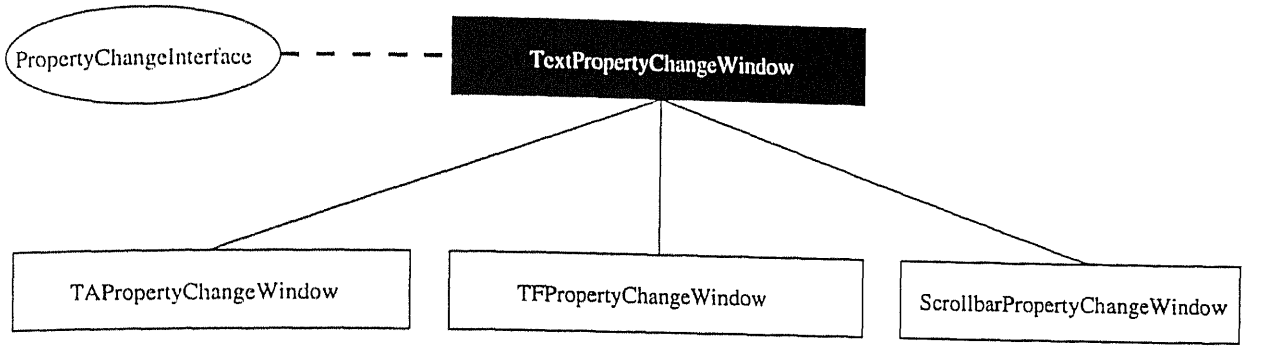


Figure 11: Hierarchy of PropertyChangeWindow classes for Text Components

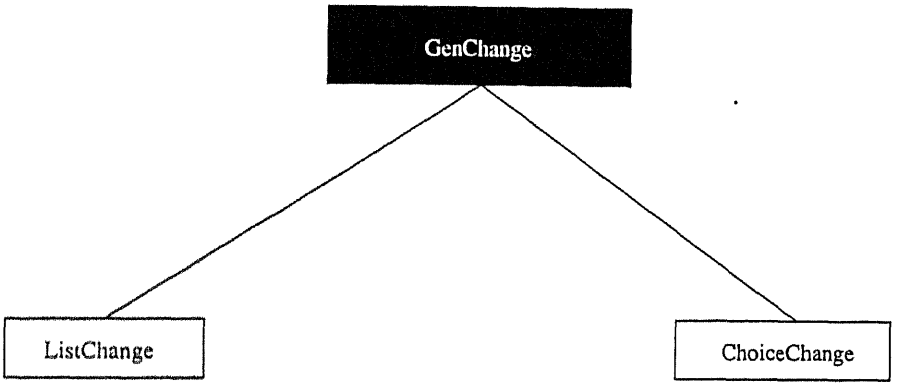


Figure 12: Hierarchy of Change classes for List and Choice

the properties of a TextArea and TextField component respectively. Figure 11 shows the relevant hierarchy. The abstract class TextPropertyChangeWindow, which is the super class of the other two classes implements the PropertyChangeInterface.

PropertyChangeWindows for List and Choice The classes ListChange and ChoiceChange allow to manipulate the order of options in the components, List and Choice respectively. They inherit from the class GenChange, where much of the functionality lies. An array storing the mapping of orders is central to this class as illustrated in Figure 5 in Chapter 3. Figure 12 shows the hierarchy.

Chapter 5

Conclusion and Suggestions for future work

5.1 Conclusion

The properties of Configurability, Customization and Dynamism for an interface were achieved using the approach implemented here. Since this approach does not introduce any additional burden to the developers, who are using the Java AWT, a tool following this approach will be readily welcomed by them.

Usage of the same set of components of Java AWT gives the following disadvantages. Since the events generated by the native implementation are not right away available to us [8], we are not able to provide direct manipulation facility to the interface for the user. This forced us to create a static copy of the interface to provide manipulation facility to the user.

Also, we are not able to provide ample alternatives for substituting a component due to the not-so-rich nature of the current AWT.

One other disadvantage which may result from the usage of Customizer is, a bit of delay will be introduced here and there. For example, at the start up time, Customizer will do some initialization work, introducing some delay. In the normal mode of operation also, every event generated has to pass through an additional layer (Customizer) introducing slight delay, although not noticeable. But, any of

these delays were not found to be annoying, so far.

5.2 Suggestions for future work

The AWT is undergoing revisions continuously. The version we used here for the implementation is JDK 1.0. The current version available is JDK 1.1.4 and still it hasn't modified its event handling mechanism to allow direct manipulation by the user. Future versions may offer better control over the events so that this tool can be modified to provide direct manipulation facility on the interface to the user.

Also, the component set might be enhanced in future versions. Then it might be possible to provide a much better set of alternatives for the substitution of components. With the current set also, one can think of introducing new components, enhancing the components set.

One other direction in which `Customizer` can be improved is to define properties for set of components, rather than for each component. ie., facilities can be provided to inherit properties of one component from another. This will enable the user to define properties for a set of components, rather than changing the properties of each individual component.

Another suggestion is to provide facility to go back to the original properties of a component, individually. Using `Customizer`, the user can only go back to the original interface as a whole. It does not provide facility to go back to the original properties for individual components.

Appendix A

Developer's Documentation

This tool helps you to make the GUI of your program dynamic, customizable and configurable . You have to design your interface as usual, but need not give undue importance for its optimal design.

To use this tool, instead of calling `show()` directly after building the GUI, call `new DynShow(container);`.

ie, if the call is like `show();`, convert it to `new Dynshow(this);`

Or if the call is like `YourContainer.show();` , convert it to

`new Dynshow(YourContainer);`

It is recommended that too much time should not be wasted on the design of the GUI, because each and every user may modify your GUI to suit their taste.

Appendix B

User Documentation

The facilities available to the users include:

- Any component can be resized.
- Any component can be repositioned.
- A component can be replaced by another.
- The label of components like Button, Label and Checkbox is modifiable.
- The color&font of components are modifiable.
- The order/names of items in a List/Choice can be changed.
- The changes made to the GUI will be retained over successive executions.
- At any point of time the user can fall back to the original interface.
- Selection of a wall paper for the GUI.
- Deletion of any component.
- Addition of static controls like Label.
- Undeletion of a deleted component

First Window

This is the GUI of the software, with two additional controls (Default Interface & Change Layout Buttons.)

Use the Default Interface Button to go back to the original interface at any point of time. Use Change Layout Button to change the present layout of your GUI.

Change Layout Window

This is a dummy and dump[1] copy of the present interface, with three additional controls(Apply, Add & Cancel Buttons).

The functions available from this window can be summarized as follows:

- To reposition a component, do drag and drop using the left mouse button.
- To resize the component do the above operation on the corners.
- Click the right mouse button over the required component to change the current properties/implementation.

  Clicking the right button within the scope of a component gives a menu with two options, Substitute and Properties. The Substitute option gives you the facility for substituting the current component with another. The Properties option gives you the provision to change the current properties of the component like color[2] ,font etc.
- To add a component of type label, use the Add Component option after pressing Add Button.
- To undelete a deleted component, use the Undelete option after pressing Add Button.
- To add or change the wallpaper of the interface, select the Wallpaper option after pressing the Add Button.

[1]The components displayed will be mere visual representations of the original components. ie., they cannot carry out any of the components functions defined.

[2]Color changes won't be reflected in Windows '95 environment for certain components

To apply any changes so far made, use the Apply button. To cancel off all the changes made so far, click Cancel button.

Properties Window

This window enables to change general properties like color font etc. and other component specific properties, depending on the component.

The changeable properties for an ordinary component (Button, label, Checkbox etc.) are label, foreground color, background color, font type, font style and font size.

For a composite control like List or Choice, the additional facility available is the provision to change the order/name of the options.

Use the Apply button to effect the changes and the Cancel button to discard all the changes made.

Appendix C

Class Hierarchy

- class java.awt.Component
  - class java.awt.Button
    - class MyButton
  - class java.awt.Container
    - class java.awt.Window
      - class java.awt.Frame
        1. class ChangeImplementation
        2. class ChangeLayout
        3. class ChangeProperties
        4. class ChoicePropertyChangeWindow (implements PropertyChangeInterface)
        5. class DynShow
        6. class GenPropertyChangeWindow (implements PropertyChangeInterface)
        7. class ButtonPropertyChangeWindow
        8. class CheckboxPropertyChangeWindow
        9. class LabelPropertyChangeWindow
        10. class ListPropertyChangeWindow (implements PropertyChangeInterface)

11. class MyFrame
12. class Showdgui
13. class dgui

* class MyWindow

- class java.awt.List
  - class MyList
- class CurrentComponent
- class DialogLayout (implements java.awt.LayoutManager)
- class FileSave
- class GenChange
- class ChoiceChange
- class ListChange
- interface PropertyChangeInterface

Appendix D

Glossary

Alternate Component A Component with which we can substitute another. The functionality may not be identical, but similar.

AWT Abstract windowing toolkit - the package containing all the classes for windowing and GUI designing.

Button A graphic component that simulates a real-life push button. When a user pushes the button, by pressing a key or a mouse button, an action takes place.

Change Implementation Substituting the functionality of component(s) with other component(s). This substitution will only affect the interface; The same set of events will be generated as before to retain the original functionality

Change Layout Changing any of the following features of a GUI: Position of component(s),size/shape of component(s),Properties of component(s) like background/foreground color,font etc, Substituting one component(s) with other components(s) etc.

Change Properties Changing the properties of component(s) like the color (Background/Foreground), Font(type,style,size), Label of component(s) like Button,Label,Checkbox etc., the Order of elements in a composite component(s) like List,Choice etc.

Checkbox It is a two-state button that can be either "on" or "off".

Choice It is a menu-like list of components, accessed by distinctive button. The user presses the button to bring up the "menu", and then chooses one of the items.

Component In the context of Java it is a base class for widgets like Button, Checkbox, List etc.

Composite Item Certain type of components like List,Choice,Menu,Menubar etc., where one or more options are listed, in contrast to simple elements like Button,Label etc.

Configurable Interface An Interface which can be configured/changed at run time.

Container In the context of Java, it is a component that can contain other AWT Components in it.

Default value A predetermined, frequently used, value for a data or control entry, intended to reduce required user entry actions.

End-user The person or group of persons who will finally interact with an application.

Event An object generated by an action by the user on a component on the GUI, which will be sent to the Event handler for that event.

Event Handler The piece of code which handles an event, generated in response to an action by the user on component(s) on the GUI.

GridBagConstraints It is used to specify constraints for Component laid out using the GridBagLayout.

GridBagLayout GridBagLayout is flexible layout manager that aligns components vertically and horizontally without requiring that the Components be the same size. Each GridBagLayout uses a rectangular grid of cells, with each component occupying one or more cells (called its display area). Each Component

managed by a GridBagLayout is associated with an instance of GridBagConstraints that specify how the Component is laid out within its display area. How a GridBagLayout places a set of Components depends on each Component's GridBagConstraints and minimum size, as well as preferred size of the Components' Container.

GridLayout It places the Components in a grid of cells. Each Component takes all the available space within its cell, and each cell is exactly the same size.

Guidelines Advice about how to design an interface.

Icon A small graphical image used to represent a window. Windows can be turned into icons or minimized to un-clutter the work space.

Interface component The elements in the interface, usually graphical, with whom the user interacts with.

Label A title or descriptor that helps a user identify displayed data. It is also a widget available in most of the toolkits.

Layout Manager A program that controls the size, placement and operation of windows on the workspace.

List A widget that provides users with a scrollable list of options from which to choose.

Menu A set of options displayed on a screen where the selection and execution of one of the options results in a change in the state of the interface.

Native toolkit The underlying native GUI library in a system, used by the Java Run time machine for the actual handling of components. ie, The actual implementation of a component will depend upon the native system library.

Pop-up Menu A Menu that appears when a particular area of screen is clicked on.

Primary window A top level window of an application.

Properties Window The window listing the current properties of the current component and gives the user facility to change them.

Pulldown Menu A Menu that is pulled down from an application's title bar.

PushButton See Button.

Scroll bar A graphical component used to change a user's view of the contents of a window.

Secondary window A child window of a primary window.

Static interface An interface where, all the components are static.

Static component A component which is static at run time ie, it's position,shape, color etc. and its implementation itself is static but accepts input from the user.

ToggleButton See Checkbox

User action Any valid action on any of the component on the interface by the end-user. For any component, zero or more actions will be defined. eg: For a `Label` component, no user action is valid where as for a `Button`, a click with the left mouse button is defined as an `ACTION_EVENT`.

Window A portion of a display screen showing particular kind of information, e.g., a command entry window, or a window for displaying the error messages.

Bibliography

[1] Simple User Interface Toolkit,
http://www.acm.org/pubs/toc/Abstracts/1046-8188/146489.html
http://nswt.tuwien.ac.at:8000/hci/readme/suit.txt
http://www.cs.virginia.edu/~suit/

[2] Sub-Arctic UI Toolkit,
http://www.cc.gatech.edu/gvu/ui/sub_arctic

[3] Empress GUI Builder,
http://www.empress.com/whatsnew/pressrel/9503a.htm
http://www.empress.com/product/optional/empgui.htm

[4] The Rise Of The Graphical User Interface, Alistair D. N. Edwards, Department of Computer Science, University of York, York, England.

http://www.msci.vt.edu/Faculty/Major/MSci3444/Gui_Rise.htm

[5] Online Seminar, HK Interface Design Inc.,
7040 W. Palmetto Park Rd. #2-256,
Boca Raton, FL.

http://expert-market.com/client/seminars/hk-sem.html

[6] In Search of a Customizable and Uniform User Interface,
Bradley M., David W. Binkley
http://info.acm.org/crossroads/xrds1-2/ecl.html

[7] HCI design guidelines,
http://www.ibm.com/ibm/hci/guidelines/guidelines.html
http://www.ibm.com/ibm/hci/resources/resource.html

[8] Tricks of the Java Programming Gurus

Glenn L Vanderburg, et al. - Samsnet 1996.

[9] Open Software Foundation. OSF/Motif Style Guide Revision 1.2 - Prentice Hall. Englewood Cliffs, New Jersey, 1993.

[10] Java: The Complete Reference
Patrick Naughton and Herbert Schildt - Tata McGraw-Hill 1997.

[11] The Java^{TM} Application Programming Interface, Volume II. James Gosling, Frank Yellin, et al.

[12] Designing user-interfaces for Java, David Reilly
http://www.inside-java.com/articles/uidesign/

[13] http://www.softis.is/technical-summary.html
http://www.softis.is/documents/LVision.html